E.PIPHANY MACROS

This appendix describes the built-in macros for SQL and operating-system commands that E.piphany supplies. You can use E.piphany macros in extraction jobs, queries against your EpiCenter datamart, and the command line for the AppServer. You can also define your own macros with EpiCenter Manager. Refer to "Macros," on page 241 for details on creating your own E.piphany macros.

BUILT-IN SQL MACROS

E.piphany provides a number of built-in SQL macros that allow you to write extraction jobs and queries against your datamart in a database-independent fashion. Where necessary, E.piphany also provides certain database-specific macros.

SQL MACRO SYNTAX

The syntax for an E.piphany SQL macro takes the following general form. Macro references begin with a pair of dollar signs. Arguments to SQL macros are enclosed within square brackets. The separator for arguments is a comma surrounded by tilde characters.

SQL MACRO SYNTAX

$$MACRO

$$MACRO[argument]

$$MACRO[arg1~,~arg2]

White space between the macro reference and the opening square bracket of the argument list is not allowed. The following example expands correctly:

```
$$NVL[MAX(col_1)~,~0]
```

The following example does *not* expand correctly:

```
$$NVL [MAX(col_1)~,~0]
```

The E.piphany macro interpreter allows white space between arguments and argument separators. However, those white-space characters are passed through and appear as part of the expanded SQL statement. Take care to ensure that any white-space characters you embed in your arguments list do not adversely affect the resulting SQL syntax.

You can determine the expanded value for most E.piphany macros by issuing the following SQL command in an EpiMeta database:

```
Select * from translation_actual
```

For usage examples, see the initialization file **templates.sql** in the following folder:

C:\Program Files\Epiphany*instance*\ConfigFiles

Replace *C* with the drive on which your E.piphany software is installed. Replace *instance* with the name of your E.piphany instance.

DATABASE-INDEPENDENT MACROS

E.piphany supports multiple database servers for EpiCenter datamarts. The E.piphany database-independent macros allow you to isolate your datamart from syntax differences that result from vendor-specific extensions to SQL.

Database-independent macros are classified into the following groups:

- Extraction macros
- EpiCenter-management macros
- General-purpose macros

Each group is discussed in a section that follows.

EXTRACTION MACROS

The extraction macros listed in Table 19 identify source-system data elements such as tables and columns, or destination data elements. The Usage column indicates the expected frequency of use for each macro.

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS (1 OF 4)

Macro	Usage	Description
$$COLUMN_CURRENT_VALUE[*table_name* ~,~ *column_name*]	Low	Expands to the value of the indicated column as of the start of the current run. Can be used to complete a "two-sided" WHERE clause that also uses the `$$COLUMN_RANGE_FILTER` macro.
$$COLUMN_FILTER[*table_name* ~,~ *column_name* ~,~ *alias_name*]	High	Expands to a "one-sided" SQL comparison expression that requires that *alias_name.column_name* be greater than or equal to the value in table *table_name* column *column_name* as of the start of the last run. This extracts "everything since last time." Although *alias_name* is optional, E.piphany suggests that you include it.

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS (2 OF 4)

Macro	Usage	Description
$$COLUMN_LAST_VALUE[*table_name* ~,~ *column_name*]	High	Expands to the value of the indicated column as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$COLUMN_RANGE_FILTER[*table_name* ~,~ *column_name* ~,~ *alias_name*]	High	Expands to a "two-sided" SQL comparison expression that requires *alias_name.column_name* to be greater than or equal to the value in table *table_name* column *column_name* as of the start of the last run, and less than or equal to this value as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."
$$DATE_CURRENT_VALUE	Low	Expands to the "current date" as of the start of the current run. This expands to a value, not to an expression, allowing you to make your own expressions. Can be used to complete a "two-sided" WHERE clause that also uses the `$$DATE_RANGE_FILTER` macro.
$$DATE_FILTER[*column_name*]	High	Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/time" as of the start of the last run. This extracts "everything since last time." No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example: `$$DATE_FILTER[oo.date_key]`

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS (3 OF 4)

Macro	Usage	Description
$$DATE_LAST_VALUE	High	Expands to the "current date" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions. This compares SQL Server datetimes. The column used for the comparison must be declared as a datetime or some variant in SQL Server. Only the date portion of the value is used; the time portion is discarded.
$$DATE_RANGE_FILTER[*column_name*]	High	Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/time" as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running." This compares SQL Server datetimes. The column used for the comparison must be declared as a datetime or some variant in SQL Server. Only the date portion of the value is used; the time portion is discarded.
$$YYYYMMDD_CURRENT_VALUE	Low	Expands to the "current date in YYYYMMDD format" as of the start of the current run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$YYYYMMDD_FILTER[*column_name*]	High	This is the same as a DATE_FILTER except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day."

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS (4 OF 4)

Macro	Usage	Description
$$YYYYMMDD_LAST_VALUE	High	Expands to the "current date in YYYYMMDD format" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$YYYYMMDD_RANGE_FILTER[*column_name*]	High	This is the same as a DATE_RANGE_FILTER except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day, but not including today."

EPICENTER-MANAGEMENT MACROS

The EpiCenter-management macros listed in Table 20 specify data elements within your datamart or conditions that might be true with respect to your datamart.

TABLE 20: EPICENTER MACROS (1 OF 2)

Macros	Usage	Description
$$CURR	High	Expands to the _A or _B suffix of the currently active tables. Allows you to reference the active or new EpiMart tables.
$$CURREXP	High	Expands to the _P or _Q suffix of the currently active backfeed tables. Allows you to reference the active or new EpiMart tables.
$$CURRHIST	High	Expands to the _X or _Y suffix of the currently active history tables. Allows you to reference the active or new EpiMart tables.

TABLE 20: EPICENTER MACROS (2 OF 2)

Macros	Usage	Description
$$CURRVIEW	High	Expands to the suffix (such as _AX or _BY) of the currently active combined view of tables and history tables. Allows you to reference the active or new EpiMart tables.
$$DEBUG	Low	If debugging in enabled: Expands to nothing if the verbosity level of the **extract.exe** command is less than 3; otherwise, returns its argument. Note: For testing that depends on whether debugging is on or off, use both DEBUG and NOT_DEBUG.
$$INITIAL_LOAD	Medium	Indicates that timestamps are to be ignored during an extraction job. (The action of this macro is not related to the Initial Load semantic type.)
$$MARTDBNAME	Medium	Returns the name of the datamart database (EpiMart).
$$METADBNAME	Medium	Returns the name of the metadata database (EpiMeta).
$$NEXT	High	Expands to the _A or _B suffix of the currently *inactive* tables.
$$NEXTEXP	High	Expands to the _P or _Q suffix of the currently *inactive* backfeed tables.
$$NEXTHIST	High	Expands to the _X or _Y suffix of the currently *inactive* history tables.
$$NOT_DEBUG	Low	If debugging is *not* enabled: Expands to a null value if the **extract** command's verbosity level is higher than 3; otherwise, returns its argument. Note: For testing that depends on whether debugging is on or off, use both DEBUG and NOT_DEBUG.
$$NOT_INITIAL_LOAD	Medium	Indicates that timestamps are to be considered during an extraction job. (The action of this macro is unrelated to the Initial Load semantic type.)

GENERAL-PURPOSE MACROS

The macros listed in Table 21 are used for a variety of purposes.

TABLE 21: GENERAL-PURPOSE SQL MACROS (1 OF 18)

Macros	Usage	Description

$$ADD_DAYS	Medium	Returns a date representation of its first argument plus its second argument as a number of days:
		For example:
		`SELECT`
		`$$ADD_DAYS[$$DBNOW~,~1]`

$$ADD_MACRO[*macro*~,~ *dbtype*~,~*value*]	Low	Defines a macro or assigns a new value. The *macro* argument is the name of the macro. The *dbtype* argument is the database server for which the value argument applies. The *value* argument is the value that the macro expands to for the indicated database server.
$$ADD_MONTHS[*date_expression*~,~ *number*]	Medium	Takes two arguments; adds the second as a number of months to the first argument, which is a date.

$$ASSERT_INDEX_EXISTS[*index_name*]	Low	Causes an SQL error if the index named in argument 1 does not exist. For example:
		`$$BEGIN_ASSERT_INDEX`
		`$$ASSERT_INDEX_EXISTS[`
		`'XPKCustMap_B]`
		`$$ASSERT_INDEX_EXISTS[`
		`'XPKAppMap_B']`
		`$$END_ASSERT_INDEX`

TABLE 21: GENERAL-PURPOSE SQL MACROS (2 OF 18)

Macros	Usage	Description
$$BATCH_PARALLEL_DEGREE	High	Sets the parallel degree for extraction jobs. The initial value is 4. You can change the value with EpiCenter Manager. See "Macros," on page 241.
$$BEGIN_ASSERT_INDEX	Low	In Oracle, declares the DECLARE INDEX_NOT_EXISTS exception. See $$ASSERT_INDEX_EXISTS.
$$BIGDATE	Low	Declares a BIGDATE data type (to record millisecond-precision timestamps).
$$BOOL_TO_YN[*testvalue*]	Low	Returns the string N if *testvalue* equals 0. Otherwise, returns Y. For example: `$$BOOL_TO_YN[6]` becomes Y.

$$CASE_BEGIN	Medium	Begins a case statement that compares an expression to a list of options and returns the value for the first matching option. Also provides a single option-value pair. For example:
		`SELECT`
		`$$CASE_BEGIN[col_name~,~opt1~,~val1]`
		`$$CASE_ELSEIF[col_name ~,~opt2~,~val2]`
		`$$CASE_ELSE[else_val]$$CASE_END`

$$CASE_ELSE	Medium	Fall-through value for a case statement that compares an expression to a list of options and returns the value for the first matching option. See CASE_BEGIN.

TABLE 21: GENERAL-PURPOSE SQL MACROS (3 OF 18)

Macros	Usage	Description
$$CASE_ELSEIF	Medium	Continues a case statement that compares an expression to a list of options and returns the value for the first matching option. See $$CASE_BEGIN.
$$CASE_END	Medium	Ends a case statement that compares an expression to a list of options and returns the value for the first matching option. See $$CASE_BEGIN.

$$CAT	High	Used as an operator to append "two" $$CAT strings. For example:
		`$$TO_CHAR[table1.col1]`
		`$$CAT '-' $$CAT`
		`$$TO_CHAR[col2]`

$$CBIN_VAL[*testval ~,~ lowerbound ~,~ upperbound ~,~ binletter*]
 	
$$CBIN_END
	Medium	Can be used to "bin" numeric values into character buckets. Multiple $$CBIN_VAL macros should be followed by a single $$CBIN_END. If testval is in the range lowerbound to upperbound (inclusive), then the expression yields binletter.
		For example:
		`$$CBIN_VAL[7~,~1~,~5~,~'A']`
		`$$CBIN_VAL[7~,~6~,~10~,~'B']`
		`$$$CBIN_END`
		
		returns the value B.

$$CHAR_1	Medium	Expands into a type definition for a single character field.

TABLE 21: GENERAL-PURPOSE SQL MACROS (4 OF 18)

Macros	Usage	Description
$$COUNT_ROWS_FROM $$COUNT_ROWS_SELECT	Low	Can be used to count the number of rows in a table. Uses sysindexes on SQL Server for fastest count. For example: `SELECT $$COUNT_ROWS_SELECT the_count $$COUNT_ROWS_FROM[MyTable]`
$$COUNTER[*initial_value*]	Low	Returns sequential row numbers for a result set, starting with *initial_value*, if supplied. For example: `SELECT $$COUNTER the_counter.`
$$CREATE_INDEX_IF_NOT_EXISTS[*index_type~,~ index_name~,~ table_name~,~ column_list~,~ after_creation_clause*]	Low	Creates an index if it is not already there. `$$DDL_BEGIN $$CREATE_INDEX_IF_NOT_EXISTS[ UNIQUE ~,~XPK_123~,~ table1~,~ss_key,iss, date_key,transtype_key,seq~ ,~] $$DDL_END`
$$DBNOW	High	Returns the date/time from the database. For example: `SELECT Col1 ss_key $$DBNOW date_modified from zork`

TABLE 21: GENERAL-PURPOSE SQL MACROS (5 OF 18)

Macros	Usage	Description

$$DDL_BEGIN	Low	Starts a block of code that changes the schema. Use outside a DECLARE block. For example:
		`$$DDL_BEGIN`
		`$$NOT_DEBUG[`
		`  $$DROP_TABLE_IF_EXISTS[`
		`  table]`
		`]`
		`$$DDL_END`

$$DDL_BEGIN_NO_DECLARE	Low	Starts a block of code that changes the schema. Use inside a DECLARE block. For example:
		`DECLARE $$VAR[txnFIXED]`
		`$$VARCHAR_50$$EOS`
		`$$DDL_BEGIN_NO_DECLARE`
		`$$VAR_ASSIGN_BEGIN[txnFIXED`
		`]`
		`SELECT`
		`$$TO_CHAR[transtype_key]`
		`$$VAR_ASSIGN_INTO[txnFIXED]`
		`FROM Transtype_O WHERE name`
		`= 'FINV_ADJUST'`
		`$$VAR_ASSIGN_END`

$$DDL_END	Low	Ends a block of SQL that changes the schema. For example:
		`$$DDL_BEGIN`
		`$$DROP_TABLE_IF_EXISTS[`
		`$$FCTTBL[ ]$$NEXT]`
		`$$DROP_TABLE_IF_EXISTS[`
		`$$FCTTBL[ ]_INC]`
		`$$DDL_END`

TABLE 21: GENERAL-PURPOSE SQL MACROS (6 OF 18)

Macros	Usage	Description

$$DDL_EXEC[*statement*]	Low	All items in the argument list are evaluated at runtime, not when the statement is parsed. This macro can construct SQL based on the values of variables computed in the same SQL block. For example:
		`$$DDL_BEGIN`
		`$$DDL_EXEC[CREATE INDEX X_table1 ON table1 ( iss, ss_key, date_key )]`
		`$$DDL_END`

$$DECLARE_BEGIN	Low	Starts a DECLARE block. For example:
		`$$DECLARE_BEGIN`
		`$$DECLARE_BODY[$$VAR[count_INC] $$EPIINT]`
		`$$DECLARE_BODY[$$VAR[count_FC] $$EPIINT]`
		`BEGIN`
		`$$VAR_ASSIGN_BEGIN[cnt_INC]`
		`SELECT COUNT(1)`
		`$$VAR_ASSIGN_INTO[cnt_INC]`
		`FROM $$FCTTBL[ ]_INC`
		`$$VAR_ASSIGN_END`

$$DECLARE_BODY[*argument*]	Low	Treats its argument as a declaration. See $$DECLARE_BEGIN.
$$DOUBLESTRING	Low	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not allowed in dimension tables.

TABLE 21: GENERAL-PURPOSE SQL MACROS (7 OF 18)

Macros	Usage	Description

$$DROP_INDEX[*table_name~,~ index_name*]	Medium	Drops the index. For example:
		`$$DDL_BEGIN`
		`$$DROP_INDEX[table1~,~ index_name]`
		`$$DDL_END`

$$DROP_TABLE_IF_EXISTS[*table_name*]	Medium	Drops the table without returning an error indicating that the table does not exist. For example:
		`$$DDL_BEGIN`
		`$$DROP_TABLE_IF_EXISTS[tbl1 ]`
		`$$DROP_TABLE_IF_EXISTS[tbl2 ]`
		`$$DDL_END`

$$ELSE	Medium	The start of the negative clause of an IF statement.
$$END_ASSERT_INDEX	Low	Ends a block of checks that indexes exist. See $$ASSERT_INDEX.
$$END_IF	Medium	Ends an IF statement, see $$IF.

$$EOS	Low	Ends an SQL statement. For example:
		`SELECT 'PROCESSED', COUNT(1), 1100 FROM table1`
		`$$EOS`

$$EPIINT	Medium	Declares an integer. For example:
		`$$DECLARE_BEGIN`
		`$$DECLARE_BODY[$$VAR[ unjoined] $$EPIINT]`
		`$$DECLARE_BODY[$$VAR[ processed] $$EPIINT]`

TABLE 21: GENERAL-PURPOSE SQL MACROS (8 OF 18)

Macros	Usage	Description
$$EPIKEY	Medium	Declares an Epiphany dimension key. See $$EPIINT.
$$EXEC_SP[*proc~,~params*]	Low	Execute a stored procedure. The *proc* argument is the procedure name. The *params* argument is a comma-separated list of parameters to the stored procedure.
$$FACTMONEY	Medium	Declares a monetary value. See $$EPIINT.
$$FACTQTY	Medium	Declares a decimal value. See $$EPIINT.
$$FLOAT	Medium	Declares a float value. See $$EPIINT.
$$IDENTITY	Medium	Declares a integer serial sequence. See $$EPIINT.

$$IF[*condition*]	Medium	Performs a conditional action. For example:
		`$$IF[$$VAR[fc_exists] = 0]`
		`$$DDL_EXEC[`
		`$$SELECT_INTO_BEGIN[tmp_tbl]`
		`SELECT`
		`*`
		`$$SELECT_INTO_BODY[tmp_tbl]`
		`FROM`
		`Old_table`
		`WHERE`
		`1=0`
		`]`
		`$$END_IF`
		`$$DDL_END`

$$IJ_FROM[*table_name1~,~table_name2*]	Low	Performs an inner join on two tables. See $$JOIN_WHERE.

TABLE 21: GENERAL-PURPOSE SQL MACROS (9 OF 18)

Macros	Usage	Description
$$INSTR[*s1~,~s2*]	Medium	Returns the position of *s1* in *s2*. For example: `$$INSTR['b' ~,~ 'abc']` returns 2.
$$INTERACTIVE_PARALLEL_DEGREE	Medium	Sets the parallel degree for ad-hoc user queries. The initial value is 4. You can change the value with EpiCenter Manager.

$$JOIN_LEFT_OUTER	Medium	Produces a condition for outer joining the first argument to the second. For example:
		`SELECT`
		`...`
		`WHERE`
		`$$JOIN_LEFT_OUTER[t1.c1 ~,~ t2.c2]`

$$JOIN_RIGHT_OUTER	Medium	Produces an equals sign appropriate for right outer joins (No arguments are needed.) For example:
		`SELECT`
		`...`
		`WHERE`
		`t1.c1 $$JOIN_RIGHT_OUTER t2.c2`

TABLE 21: GENERAL-PURPOSE SQL MACROS (10 OF 18)

Macros	Usage	Description
$$JOIN_WHERE[*join_condition*]	Low	Supplies the WHERE clause for a join. For example: `SELECT col1, col2 FROM Table1 s $$LOJ_FROM[table2 m~,~s.iss = m.iss AND s.col2=m.col2] $$LOJ_FROM[table3 d~,~m.col1 = d.col1] WHERE 1=1 $$JOIN_WHERE[m.col1=d.col1(+)] $$JOIN_WHERE[s.iss = m.iss (+) AND s.col2 = m.col2 (+)]`
$$LENGTH[s]	Medium	Returns the length of a string. For example: `$$LENGTH['abc']` returns 3.
$$LOJ_FROM[*join_condition*]	Low	Performs a left outer join. See $$JOIN_WHERE.
$$LONGSTRING	Low	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not allowed in dimension tables.
$$MAX_SYS_DATE	Medium	Returns the highest date supported by the database.
$$METADBNAME	Medium	Returns the name of the current metadata database (SQL Server) or schema (Oracle).

TABLE 21: GENERAL-PURPOSE SQL MACROS (11 OF 18)

Macros	Usage	Description
$$MODULO[*x* ~, ~*y*]	Low	Returns the remainder when *x* is divided by *y*. For example: `MODULO{7~,~4]` returns 3.
$$MONEYSTRING	Low	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not allowed in dimension tables.

$$NBIN_VAL[*testval~,~ lowerbound ~,~ upperbound ~,~ binnumber*]
$$NBIN_END
	Medium	Can be used to "bin" numeric values into numeric buckets. Multiple $$NBIN_VAL macros should be followed by a single $$NBIN_END. If testval is in the range lowerbound to upperbound (inclusive), then the expression yields binnumber.
		For example:
		`$$NBIN_VAL[7~,~1~,~5~,~1]`
		`$$NBIN_VAL[7~,~6~,~10~,~ 2]`
		`$$NBIN_END`
		return the value 2.

$$NO_FROM_LIST	Medium	Supplies the "dummy" FROM clause needed by some database vendors. For example:
		`SELECT 'MODIFIED',`
		`$$VAR[modified], 1050`
		`$$NO_FROM_LIST$$EOS`

$$NUMBER(9,5)	Medium	Declares a decimal(9,5). See $$EPIINT.

TABLE 21: GENERAL-PURPOSE SQL MACROS (12 OF 18)

Macros	Usage	Description
$$NVL[*expression~,~value*]	High	When the first argument is NULL, replace it with the value in the second argument. For example: `SELECT $$TO_CHAR[$$NVL[MAX(col1)~,~1]]`
$$ORACLE[*expression*]	High	Expands to nothing if the database is not Oracle. For example: `SELECT COUNT(1) FROM $$SQLSERVER[sysobjects]$$ORACLE[tabs]`
$$RAISE_EXCEPTION[*exception*]	Low	Raises the given exception (as a variable on Oracle, as a string on SQL Server). For example: `$$RAISE_EXCEPTION[MyException]` raises an exception.
$$REMOVE_MACRO[*macro*]	Low	Removes the definition of the macro indicated by the *macro* argument.
$$RENAME_OBJECT	Medium	Renamed tables or other database objects. For example: `$$RENAME_OBJECT[oldtablename ~,~newtblname]`

TABLE 21: GENERAL-PURPOSE SQL MACROS (13 OF 18)

Macros	Usage	Description

$$SELECT_INTO_BEGIN[*table_name*]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. For example:
		`$$SELECT_INTO_BEGIN[temp_tab]`
		`SELECT`
		`*`
		`$$SELECT_INTO_BODY[temp_tab]`
		`FROM`
		`Old_tab`
		`WHERE`
		`1=0`
		(See "Oracle Macros," on page 407 for more additional similar macros.)

$$SELECT_INTO_BODY[*table_name*]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. See $$SELECT_INTO_BEGIN.
$$SMALLDATE	Medium	Declares a SMALLDATETIME. See $$EPIINT.
$$SMALLINT	Medium	Declares a double-byte integer. See $$EPIINT.

$$SQLSERVER[*expression*]	High	Expands into nothing if the database engine is not SQL Server. For example:
		`SELECT COUNT(1)`
		`FROM`
		`$$SQLSERVER[sysobjects]$$ORACLE[tabs]`

$$SSKEY	Low	Declares the type Epiphany uses for **ss_keys**. See $$EPIINT.

TABLE 21: GENERAL-PURPOSE SQL MACROS (14 OF 18)

Macros	Usage	Description
$$SUBSTRING[*expression~,~ start~,~length*]	Medium	Performs a substring operation. For example: `$$SUBSTRING[name~,~1~,~8]`
$$SUPERNVL	Medium	Converts a null value for the first argument into the second argument. The resulting column is not nullable in the schema definition of the result set. For example: `SELECT $$SUPERNVL[col1~,~'UNKNOWN']`
$$TABLE_EXISTS_CONDITION[*table_name*]	Low	Detects if a table exists. For example: `SELECT COUNT(1) FROM $$SQLSERVER[sysobjects]$$ORACLE[tabs] WHERE $$TABLE_EXISTS_CONDITION[table_name]`
$$TABLE_WITH_PREFIX[*database_name~,~ table_name*]	Medium	Qualifies a table name with a database or user name. For example: `SELECT source_system_key iss FROM $$TABLE_WITH_PREFIX[ $$METADBNAME~,~ source_system]`
$$TINYINT	Medium	Declares a single-byte integer. See $$EPIINT.

TABLE 21: GENERAL-PURPOSE SQL MACROS (15 OF 18)

Macros	Usage	Description

$$TO_CHAR[*expression*]	High	Converts a value to a character string. In Oracle, the value must be numeric. For example:
		`SELECT`
		`$$TO_CHAR[$$NVL[MAX(col1)]]`
		As an alternative, you can use the $$TO_CHAR_UNIVERSAL macro, which accepts both numeric and nonnumeric values.

$$TO_CHAR_UNIVERSAL[*expression*]	Medium	Converts a value to a character string.
$$TO_DATE[*expression*]	Medium	Converts a value to a database date with the following format: *MM/DD/YYYY HH24:MM:SS* *MM* is the month in two-digit notation, DD is the two-digit day, *YYYY* is the year, *HH24* is the hour in 24-hour notation, *MM* is the two-digit minute, and *SS* is the two-digit second.
$$TO_DATEFMT[*expr~, ~format*]	Low	Converts expression to a date type with the appropriate Oracle format (*format* is ignored on SQL Server).

$$TO_EPIDATE[*expression*]	High	Converts a date to the string format preferred by EpiChannel. This macro should be used for all columns that have physical type SMALLDATE. For example:
		`Select col1 ss_key,`
		`$$TO_EPIDATE[date_col]`
		`date_modified`
		`from zork`

TABLE 21: GENERAL-PURPOSE SQL MACROS (16 OF 18)

Macros	Usage	Description
$$TO_HHMMSS[*datetime_var*]	Medium	Converts a datetime variable to a string of the form: *HHMMSS* where *HH* represents the hour, *MM* represents the minute, and *SS* represents the second.
$$TO_INT[*expression*]	Medium	Converts *expression* to an integer type.
$$TO_NUMBER[*expression*]	Medium	Converts an expression to a number. For example: `$$TO_NUMBER['123']` returns `123`.
$$TO_TIME[*expression*]	Medium	Converts its argument to a time representation. For example: `SELECT $$TO_TIME[$$DBNOW]`
$$TO_YYYYMMDD[*expression*]	Medium	Converts a data to a `YYYYMMDD` string.

$$TRANSLATE_BEGIN
$$TRANSLATE_VAL[*expression*~,~
searchval~,~
translationval~,~
nestedcall~,~
$$TRANSLATE_ELSE[*otherterm*]]
$$TRANSLATE_ELSE
$$TRANSLATE_END
	Medium	Searches expression for occurrences of each *searchval* and returns the *translationval*. Note that $$TRANSLATE_VAL terms should be nested inside of each other. An optional $$TRANSLATE_ELSE can be nested inside the final $$TRANSLATE_VAL. For example:
		`$$TRANSLATE_BEGIN`
		`$$TRANSLATE_VAL['abcdef'~,~`
		`bce~,~'Value1'~,~`
		`$$TRANSLATE_VAL['abcdef'~,~`
		`cde~,~ 'Value2'~,~`
		`$$TRANSLATE_ELSE['Other']]]`
		`$$TRANSLATE_END`
		returns `Value2`.

TABLE 21: GENERAL-PURPOSE SQL MACROS (17 OF 18)

Macros	Usage	Description
$$TRANSTYPE[*name*]	High	Returns the transtype number corresponding to the name that is the first argument of the macro.
$$UNKNOWN_DATE	Medium	Expands to a date of the form *MM/DD/YYYY* (`01/01/1990` by default). You can set a new value in this same format with EpiCenter Manager. See "Macros," on page 241.
$$VAR[*variable_name*]	Medium	References a database variable. For example: `SELECT 'PROCESSED', $$VAR[processed], 1100 $$NO_FROM_LIST$$EOS`
$$VAR_ASSIGN_BEGIN[*variable_name*]	Medium	Assigns to a database variable. For example: `$$VAR_ASSIGN_BEGIN[max_key] SELECT $$TO_CHAR[$$NVL[ MAX(col1)~,~1]] $$VAR_ASSIGN_INTO[max_key] FROM table2 $$VAR_ASSIGN_END`
$$VAR_ASSIGN_END	Medium	Assigns to a database variable. See $$VAR_ASSIGN_BEGIN.
$$VAR_ASSIGN_INTO[*variable_name*]	Medium	Assigns to a database variable. See $$VAR_ASSIGN_BEGIN.
$$VARCHAR_5	Medium	Declares a variable-width character datatype that holds a maximum of 5 characters. See $$EPIINT.
$$VARCHAR_15	Medium	Declares a variable-width character datatype that holds a maximum of 15 characters. See $$EPIINT.

TABLE 21: GENERAL-PURPOSE SQL MACROS (18 OF 18)

Macros	Usage	Description
$$VARCHAR_25	Medium	Declares a variable-width character datatype that holds a maximum of 25 characters. See $$EPIINT.
$$VARCHAR_50	Medium	Declares a variable-width character datatype that holds a maximum of 50 characters. See $$EPIINT.
$$VARCHAR_100	Medium	Declares a variable-width character datatype that holds a maximum of 100 characters. See $$EPIINT.
$$VARCHAR_255	Medium	Declares a variable-width character datatype that holds a maximum of 255characters. See $$EPIINT.

DATABASE-SPECIFIC SQL MACROS

This section lists macros and discusses concerns that apply to specific database servers.

ORACLE MACROS

Oracle-specific SQL macros control the physical characteristics of an Oracle EpiCenter. Oracle tables are stored in a logical entity called a tablespace. Because of the various size requirements of EpiCenter objects, such as fact tables and indexes, dimension tables and indexes, EpiCenter allows you to configure which tablespace is used for each object type. When appropriate, the expansions of Oracle-specific SQL macros refer to tablespace names.

TABLE 22: ORACLE-SPECIFIC SQL MACROS

Macro	Description
$$ANALYZE_TABLE[*table_name*]	Performs a size-based analysis of the indicated table.
$$DIMINDEX_TABLESPACE	Used for dimension indexes (including those on aggregates and mini-dimensions).
$$DIMTABLESPACE	Used for dimension tables (including aggregates and mini-dimensions).
$$FACTINDEX_TABLESPACE	Used for fact indexes (including those on aggregates and clusters).
$$FACTTABLESPACE	Used for fact tables (including aggregates and clusters, as well as temporary objects used during semantics).
$$METATABLESPACE	Expands to the name of the tablespace to use for metadata tables on your system.
$$SELECT_INTO_BEGIN_OPT[*table_name, option_string*]	Expands to a CREATE TABLE statement. The *option_string* argument passes a list of options to the resulting statement.
$$SELECT_INTO_BEGIN_TS[*table_name, tablespace*]	Expands to a CREATE TABLE statement. The *tablespace* argument passes the name of the desired tablespace for the new table to the resulting statement.
$$TEMP_TABLESPACE	Used for all Application Server temporary tables (those needed for query post-processing at runtime).

Normally, the values for these macros are set to match the names of the tablespaces as they are created by the Oracle initialization script provided by E.piphany as described in the *E.piphany e.4 Installation Guide*.

To use alternate names for these tablespaces, you need to run SQL statements such as the following against your EpiMeta database.

```
update translation_Actual set actual_string =
'your_tablespace_name' where store_type = 'Oracle' and
translation_string = 'FACTTABLESPACE'
```

SQL SERVER DATA TYPES AND E.PIPHANY MACROS

Fact semantic types that use aggregation operators, such as SUM(), are sensitive to decimal data types such as `NUMBER(x,y)`. When new tables are created with SELECT INTO statements based on aggregates of these data types, columns that use those data types expand to hold the largest decimal value. This expansion can unnecessarily increase the overall size of the fact table. For this reason, both `FACTQTY` and `FACTMONEY` map to the `MONEY` data type in SQL Server.

SQL SERVER MACROS

The following macros apply to SQL Server.

TABLE 23: SQL SERVER MACROS

Macro	Usage	Description
$$TIMESTAMP_FILTER[*column_name*]	High	Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run. This extracts "everything since last time." This compares timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type, not as a time or date.
$$TIMESTAMP_LAST_VALUE	High	Expands to the "current timestamp" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
$$TIMESTAMP_RANGE_FILTER[*column_name*]	High	Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running." This compares timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type, not as a time or date.

SYSTEM-CALL MACROS

E.piphany system-call macros allow you to encapsulate operating-system commands in extraction jobs. You typically use system-call macros to pass filenames or user names that have been stored in metadata to operating-system commands. Many commands, such as **RENAME** and **DIR**, are unable to read a database, and few commands can read E.piphany-created structures.

SYSTEM-CALL MACRO SYNTAX

$$*MACRO*

$$*MACRO*[*argument*]

$$*MACRO*[*arg1, arg2*]

Names specific data stores can be abstracted within the job definition through the use of data-store roles. See "System Calls," on page 79 for more information about roles. Unless stated otherwise, the arguments for a macro are the names of data-store roles that have been defined for an extraction job.

You can use multiple arguments in most cases, which results in an expansion of each argument. Note that the expansion of both of the following macros is the same:

```
$$macro[arg1, arg2]

$$macro[arg1] $$macro[arg2]
```

If a macro reference does not match the name of a macro that has been defined, the extraction job halts with an error. If the argument to a macro is a role name, and the job does not define that role, the extraction job halts with an error.

Table 24, on page 412 lists the E.piphany system-call macros.

TABLE 24: E.PIPHANY SYSTEM-CALL MACROS (1 OF 4)

Macros	Purpose	Usage	Description
$$AGG	Child Procs	High	The name of the AggBuilder executable in $$EPIBIN. For example: `$$AGG $$EXC_ARGS -j $$JOB_NAME`
$$APPSERVERHOST	Registry	Low	The value of this Registry variable.
$$APPSERVERPORT	Registry	Low	The value of this Registry variable.
$$CHARTSLOGFILE	Registry	Low	The value of this Registry variable.
$$CHARTSOUTPUTDIR	Registry	Low	The value of this Registry variable.
$$DATABASE[*role*]	Database Login	High	Translates to the name of the database or instance when the data-store role is associated with a specific database server (that is, not a data store of type ODBC or File). For example: `isql /S $$SERVER[MyRole] /U $$USER[MyRole] /P $$PASSWORD[MyRole] /d $$DATABASE[MyRole] /w 300 /i /Q "gen_tests_run"`
$$DBVENDOR[*role*]	Database Login	Medium	Translates to the vendor of the database server associated with a data-store role.
$$DEBUG_LEVEL	Command Line	Low	Translates to the current verbosity level of EpiChannel. Use this to pass EpiChannel's verbosity onto the subprocesses it spawns via system calls.

TABLE 24: E.PIPHANY SYSTEM-CALL MACROS (2 OF 4)

Macros	Purpose	Usage	Description
$$DIRNAME[*role*]	File ID	Medium	Translates to the directory name of the data store associated with *role*, without the last filename component and without a trailing slash. If the role is **WorkingDir**, the directory name's last component is a unique subdirectory generated for this particular run of EpiChannel. For example: `echo "DIRNAME is " $$DIRNAME[WorkingDir]`
$$DSN	Database Login	Medium	Translates to the ODBC connection string for the database. This string may be generated even for databases accessed using native APIs.
$$EPIBIN	Child Procs	Medium	The name of the **Win32** directory under the **InstanceRootDir** Registry variable.
$$EXC	Child Procs	High	The name of the **extract.exe** executable in $$EPIBIN.
$$EXC_ARGS	Child Procs	High	The recognized portions of the **extract.exe** command line.
$$EXC_CMD	Child Procs	Medium	The **extract.exe** program and its arguments other than job name. You can use this to fire subsidiary runs. For example: `$$EXC_CMD -j performance`

TABLE 24: E.PIPHANY SYSTEM-CALL MACROS (3 OF 4)

Macros	Purpose	Usage	Description
$$FILENAME[*role*]	File ID	Medium	Translates to the filename of the data store associated with *role*. If the role is **WorkingDir**, the file name is the name of the EpiChannel log file. For example: `echo "FILENAME is " $$FILENAME[Working Dir]`
$$INSTANCE_NAME	Registry	Medium	The name of the instance's Registry subtree.
$$INSTANCEROOTDIR	Child Procs	Low	Value of the **InstanceRootDir** Registry variable.
$$JOB_NAME	Child Procs	High	The name of the current job.
$$PASSWORD[*role*]	Database Login	High	The password for the user of the database-server or host associated with the indicated data-store role.
$$PATH[*role*]	File ID	Medium	Translates to the full pathname of the file that is associated with *role*.
$$PROGRAM_NAME	Child Procs	Low	The name of the current **extract.exe** program.
$$REGISTRY_EPIPATH	Registry	Low	Name of the E.piphany Registry key.
$$SERVER[*role*] $$SQLNET[*role*]	Database Login	High	Translates to the name of the database server for the data store that is associated with *role*. For SQL Server, this is the hostname of the computer on which the database server resides. For Oracle, this is the SQLNet ID (SID). SERVER and SQLNET are identical in behavior and can be interchanged. The data-store type must be a specific database server (not ODBC or File.)

TABLE 24: E.PIPHANY SYSTEM-CALL MACROS (4 OF 4)

Macros	Purpose	Usage	Description
$$USER	Database Login	High	The user name for the database-server or host associated with the indicated data-store role.
$$VERSION[*role*]	Database Login	Low	Translates to the version number (or string) of the database server that is associated with *role*. The data-store type must be a specific database server (not ODBC or File).